More Proofs By Induction
(Trees and General Datatypes)

COS 326
Andrew W. Appel
Princeton University


slides copyright 2013-2015 David Walker and Andrew W. Appel
Theorem: For all lists \( xs \), property\((xs)\).

Proof: By induction on lists \( xs \).

**Case:** \( xs == [ ] \):
... no uses of IH ...

**Case:** \( xs == \text{hd} :: \text{tl} \):
IH: property\((\text{tl})\)
Theorem: For all lists $xs$, property($xs$).
Proof: By induction on lists $xs$.

Case: $xs == []$: 
... no uses of IH ...

Case: $xs == hd :: tl$: 
IH: property($tl$)

one case for empty list

one case for nonempty lists
IH may be used on smaller lists

In general, cases must cover all the lists:
• other possibilities: case for [], case for $x1::[]$, case for $x1::x2::tl$
Theorem: For all lists $xs$, property($xs$).

Proof: By induction on lists $xs$.

Case: $xs == []$:
   ... no uses of IH ...

Case: $xs == \text{hd} :: \text{tl}$:
   IH: property($\text{tl}$)

one case for empty list

one case for non-empty lists
IH may be used on smaller lists

In general, cases must cover all the lists:

- other possibilities: case for $[]$, case for $x1::[]$, case for $x1::x2::\text{tl}$

just splitting the case for non-empty lists in 2 again
More General Template for Inductive Datatypes

type t = C1 of t1 | C2 of t2 | ... | Cn of tn

types t1, t2 ... tn, may contain 1 or more occurrence of t within them.

Examples:

type mylist =
  MyNil
  | MyCons of int * mylist

type 'a tree =
  Leaf
  | Node of 'a * 'a tree * 'a tree

recursive occurrences
More General Template for Inductive Datatypes

**Theorem:** For all $x : t$, property(x).

**Proof:** By induction on structure of values $x$ with type $t$. 
More General Template for Inductive Datatypes

Theorem: For all $x : t$, property($x$).

Proof: By induction on structure of values $x$ with type $t$.

Case: $x == \text{C1 v:}$

... use IH on components of $v$ that have type $t$ ...

Case: $x == \text{C2 v:}$

... use IH on components of $v$ that have type $t$ ...

Case: $x == \text{Cn v:}$

... use IH on components of $v$ that have type $t$ ...
A PROOF ABOUT TREES
Another example

type 'a tree = Leaf | Node of 'a * 'a tree * 'a tree

let rec tm f t =
  match t with
  | Leaf -> Leaf
  | Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<>) f g =
  fun x -> f (g x)
Another example

Theorem:
For all (total) functions \( f : b \rightarrow c \),
For all (total) functions \( g : a \rightarrow b \),
For all trees \( t : \text{a tree} \),
\( \text{tm } f \ (\text{tm } g \ t) = \text{tm } (f <> g) \ t \)

```ocaml
type 'a tree = Leaf | Node of 'a * 'a tree * 'a tree

let rec tm f t =
  match t with
  | Leaf -> Leaf
  | Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<>) f g =
  fun x -> f (g x)
```
Theorem:
For all (total) functions \( f : b \rightarrow c \),
For all (total) functions \( g : a \rightarrow b \),
For all trees \( t : a \) tree,
\( \text{tm } f \ (\text{tm } g \ t) =: \text{tm } (f <> g) \ t \)

To begin, let’s pick an arbitrary total function \( f \) and total function \( g \).
We’ll prove the theorem without assuming any particular properties of \( f \) or \( g \) (other than the fact that the types match up). So, for the \( f \) and \( g \) we picked, we’ll prove:

Theorem:
For all trees \( t : a \) tree,
\( \text{tm } f \ (\text{tm } g \ t) =: \text{tm } (f <> g) \ t \)
Theorem:
For all trees t : a tree,
\(tm f \ (tm g \ t) == tm \ (f <> g) \ t\)

```
let rec tm f t =
  match t with
  | Leaf -> Leaf
  | Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<> f g =
  fun x -> f (g x)
```
Another example

Theorem:
For all trees t : a tree,
\[ \text{tm } f \ (\text{tm } g \ t) = \text{tm } (f <> g) \ t \]

let rec tm f t =
  match t with
  | Leaf -> Leaf
  | Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<>) f g =
  fun x -> f (g x)

Case: \( t = \text{Leaf} \)

No inductive hypothesis to use.
(Leaf doesn’t contain any smaller components with type tree.)

Proof:
\[
\begin{align*}
  \text{tm } f \ (\text{tm } g \ \text{Leaf}) \\
  = \text{tm } f \ \text{Leaf} \quad \text{(eval)} \\
  = \text{Leaf} \quad \text{(eval)} \\
  = \text{tm } (f <> g) \ \text{Leaf} \quad \text{(reverse eval)}
\end{align*}
\]
Theorem:
For all trees $t: a$ tree,
$tm f (tm g t) == tm (f <> g) t$

Case: $t = Node(v, l, r)$

IH1: $tm f (tm g l) == tm (f <> g) l$
IH2: $tm f (tm g r) == tm (f <> g) r$

let rec tm f t =
  match t with
    | Leaf -> Leaf
    | Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<> f g =
  fun x -> f (g x)
Another example

Theorem:
For all trees t : a tree,
\( \text{tm } f \text{ (tm } g \text{ t) } == \text{tm } (f <> g) \text{ t} \)

Case: \( t = \text{Node}(v, l, r) \)

IH1: tm f (tm g l) == tm (f <> g) l
IH2: tm f (tm g r) == tm (f <> g) r

Proof:
\[
\text{tm } f \text{ (tm } g \text{ (Node } (v, l, r))) == \text{tm (f <> g) (Node } (v, l, r))
\]

let rec tm f t =
  match t with
  | Leaf -> Leaf
  | Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<> ) f g =
  fun x -> f (g x)
Another example

Theorem:
For all trees \( t \) : a tree,
\[
\text{tm} f (\text{tm} g t) = \text{tm} (f <> g) t
\]

Case: \( t = \text{Node}(v, l, r) \)

IH1: \( \text{tm} f (\text{tm} g l) = \text{tm} (f <> g) l \)
IH2: \( \text{tm} f (\text{tm} g r) = \text{tm} (f <> g) r \)

Proof:
\[
\text{tm} f (\text{tm} g (\text{Node} (v, l, r))) = \text{tm} f (\text{Node} (g v, \text{tm} g l, \text{tm} g r)) \\
= \text{tm} (f <> g) (\text{Node} (v, l, r))
\]

let rec tm f t =
match t with
| Leaf -> Leaf
| Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<>)) f g =
fun x -> f (g x)
Another example

**Theorem:**
For all trees $t : a$ tree,
$TM(f) (TM(g) t) = TM(f <> g) t$

**Case:** $t = Node(v, l, r)$

IH1: $TM(f) (TM(g) l) = TM(f <> g) l$
IH2: $TM(f) (TM(g) r) = TM(f <> g) r$

**Proof:**

$TM(f) (TM(g) (Node (v, l, r)))$

$= TM(f) (Node (g v, TM(g) l, TM(g) r))$  \hspace{1cm} (eval inner TM)

Node ($(f <> g) v, TM(f <> g) l, TM(f <> g) r)$

$= TM(f <> g) (Node (v, l, r))$  \hspace{1cm} (eval reverse)

```
let rec tm f t = 
    match t with
    | Leaf -> Leaf
    | Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<> f g = 
    fun x -> f (g x)
```
Another example

Theorem:
For all trees t : a tree,
\( \text{tm } f \ (\text{tm } g \ t) = \text{tm } (f \ <> \ g) \ t \)

Case: \( t = \text{Node}(v, l, r) \)

IH1: \( \text{tm } f \ (\text{tm } g \ l) = \text{tm } (f \ <> \ g) \ l \)
IH2: \( \text{tm } f \ (\text{tm } g \ r) = \text{tm } (f \ <> \ g) \ r \)

Proof:
\[
\begin{align*}
\text{tm } f \ (\text{tm } g \ (\text{Node}(v, l, r))) \\
= \text{tm } f \ (\text{Node}(g v, \text{tm } g l, \text{tm } g r)) \\
= \text{Node}(f (g v), \text{tm } f \ (\text{tm } g l), \text{tm } f \ (\text{tm } g r)) \\
\end{align*}
\]

(\text{eval inner } \text{tm})

(\text{eval – since } g, \text{tm} \text{ are total})

\[
\begin{align*}
\text{Node} \ ((f \ <> \ g) v, \ (f \ <> \ g) l, \ (f \ <> \ g) r) \\
= \text{tm } (f \ <> \ g) \ (\text{Node}(v, l, r)) \\
\end{align*}
\]

(\text{eval reverse})
Another example

Theorem:
For all trees $t : a$ tree,
$tm f (tm g t) == tm (f <> g) t$

Case: $t = Node(v, l, r)$

IH1: $tm f (tm g l) == tm (f <> g) l$
IH2: $tm f (tm g r) == tm (f <> g) r$

Proof:
$$tm f (tm g (Node (v, l, r)))$$
$$== tm f (Node (g v, tm g l, tm g r))$$
$$== Node (f (g v), tm f (tm g l), tm f (tm g r))$$
$$== Node ((f <> g) v, tm (f <> g) l, tm f (tm g r))$$
$$== Node ((f <> g) v, tm (f <> g) l, tm (f <> g) r)$$
$$== tm (f <> g) (Node (v, l, r))$$

let rec tm f t =
match t with
  | Leaf -> Leaf
  | Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<>') f g =
fun x -> f (g x)
Another example

Theorem:
For all trees t : a tree,
tm f (tm g t) == tm (f <> g) t

Case: t = Node(v, l, r)

IH1: tm f (tm g l) == tm (f <> g) l
IH2: tm f (tm g r) == tm (f <> g) r

Proof:
\[ \text{tm f (tm g (Node (v, l, r)))} \]
\[ == \text{tm f (Node (g v, tm g l, tm g r))} \]
\[ == \text{Node (f (g v), tm f (tm g l), tm f (tm g r))} \]
\[ == \text{Node ((f <> g) v, tm f (tm g l), tm f (tm g r))} \]
\[ == \text{Node ((f <> g) v, tm (f <> g) l, tm f (tm g r))} \]
\[ == \text{Node ((f <> g) v, tm (f <> g) l, tm (f <> g) r)} \]
\[ == \text{tm (f <> g) (Node (v, l, r))} \]

let rec tm f t =
    match t with
    | Leaf -> Leaf
    | Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<>) f g =
    fun x -> f (g x)
Theorem:
For all trees \( t \) : a tree,
\[
\text{tm } f \ (\text{tm } g \ t) = \text{tm } (f \ <> \ g) \ t
\]

Case: \( t = \text{Node}(v, l, r) \)

IH1: \( \text{tm } f \ (\text{tm } g \ l) = \text{tm } (f \ <> \ g) \ l \)
IH2: \( \text{tm } f \ (\text{tm } g \ r) = \text{tm } (f \ <> \ g) \ r \)

Proof:
\[
\begin{align*}
\text{tm } f \ (\text{tm } g \ (\text{Node}(v, l, r))) & \equiv \text{tm } f \ (\text{Node}(g \ v, \text{tm } g \ l, \text{tm } g \ r)) \\
& \equiv \text{Node}(f \ (g \ v), \text{tm } f \ (\text{tm } g \ l), \text{tm } f \ (\text{tm } g \ r)) \\
& \equiv \text{Node}((f \ <> \ g) \ v, \text{tm } f \ (\text{tm } g \ l), \text{tm } f \ (\text{tm } g \ r)) \\
& \equiv \text{Node}((f \ <> \ g) \ v, \text{tm } (f \ <> \ g) \ l, \text{tm } (f \ <> \ g) \ r) \\
& \equiv \text{tm } (f \ <> \ g) \ (\text{Node}(v, l, r))
\end{align*}
\]

let rec tm f t =
match t with
| Leaf -> Leaf
| Node (x, l, r) -> Node (f x, tm f l, tm f r)

let (<> ) f g =
fun x -> f (g x)
Theorem: For all x : ‘a tree, property(x).

Proof: By induction on the structure of trees x.

Case: x == Leaf:

... no use of inductive hypothesis (this is the smallest tree) ...

Case: x == Node (v, left, right):

IH1: property(left)  
IH2: property(right)

... use IH1 and IH 2 in your proof ...
Summary of Template for Inductive Datatypes

Theorem: For all x : t, property(x).

Proof: By induction on structure of values x with type t.

Case: x == C1 v:

... use IH on components of v that have type t ...

Case: x == C2 v:

... use IH on components of v that have type t ...

Case: x == Cn v:

... use IH on components of v that have type t ...

type t = C1 of t1 | C2 of t2 | ... | Cn of tn
Exercise

type 'a tree = Leaf of 'a | Node of 'a tree * 'a tree

let rec flip (t: 'a tree) =
  match t with
  | Leaf _ -> t
  | Leaf _ -> t
  | Node (a,b) -> Node (flip b, flip a)

Theorem: flip(flip t) = t.
Exercise

type 'a tree = Leaf of 'a | Node 'a tree * 'a tree

let rec flip (t: 'a tree) =
  match t with
  | Leaf _ -> t
  | Leaf _ -> flip t
  | Node (a,b) -> Node (flip b, flip a)

Theorem:  flip(flip t) = t.

Theorem:  flip(flip (flip t)) = flip t.